Check GPU

In [1]:
```python
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use TensorFlow version 1.0 or newer.  You are us
ing {}'.format(tf.__version__)
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please ensure you have installed TensorFlow correctly')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
TensorFlow Version: 2.6.0
Default GPU Device: /device:GPU:0
Num GPUs Available:  1
```

In [2]:
```python
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Out[2]:
```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 17875864096839324546,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 9898950656
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 8683298867691047183
 physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 3080 Ti, pci bus id: 0000:01:00.0, compute capability: 8.
6"]
```

Import Libraries

In [3]:
```python
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout
from tensorflow.keras.losses import mean_absolute_error, mean_squared_error
from tensorflow.keras.optimizers import Nadam, Adam
from tensorflow.keras.initializers import lecun_normal

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
```

California Housing Dataset

https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

In [4]:
```python
df_train = pd.read_csv("california_housing_train.csv")
df_test = pd.read_csv("california_housing_test.csv")
```

The dataset is splitted into 8.5:1.5 (Train:Test) ratio.

In [44]:
```python
print(len(df_train))
print(len(df_test))
```

```
17000
3000
```

In [5]:
```python
df_train.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Drop features accordingly and choose the target feature

In [6]:
```python
df_train = df_train.drop(["longitude", "latitude", "housing_median_age","households","median_income","total_rooms",
"total_bedrooms"], axis=1)
df_test = df_test.drop(["longitude", "latitude", "housing_median_age","households","median_income","total_rooms", "t
otal_bedrooms"], axis=1)
```

In [7]:
```python
from sklearn.preprocessing import StandardScaler

X_train = df_train.drop([ "median_house_value",], axis=1).values
y_train = df_train.median_house_value.values
X_test = df_test.drop(["median_house_value",], axis=1).values
y_test = df_test.median_house_value.values
```

In [8]:
```python
scaler = StandardScaler()

X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)
```

Using Callbacks to avoid using too many unnecessary epochs

Change the value of 'patience' accordingly, for the model to check if the evaluation metrics is not improving

In [9]:
```python
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
```

Not Using Monte Carlo Dropout

Lightweight Model (Regular)

In [10]:
```python
model_NN_light = Sequential([InputLayer(input_shape=(1,)),
            Dropout(0.4),
            Dense(100, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(200, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(200, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(100, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(1, activation=None)
            ])
print(model_NN_light)
```

```
<keras.engine.sequential.Sequential object at 0x00000215E875F3D0>
```

In [11]:
```python
model_NN_light.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, 1)                 0
_________________________________________________________________
dense (Dense)                (None, 100)               200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_2 (Dense)              (None, 200)               40200
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               20100
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
Total params: 80,801
Trainable params: 80,801
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```python
model_NN_light.compile(loss="mean_squared_error",
              optimizer="Nadam",
              metrics=["mean_squared_error"])
```

In [13]:
```python
history_NN_light = model_NN_light.fit(X_train_norm, y_train, epochs=201, validation_split=0.2, callbacks=[callback])
```

```
Epoch 1/201
425/425 [==============================] - 4s 7ms/step - loss: 29605449728.0000 - mean_squared_error: 29605449728.000
0 - val_loss: 24009990144.0000 - val_mean_squared_error: 24009990144.0000
Epoch 2/201
425/425 [==============================] - 4s 9ms/step - loss: 15972195328.0000 - mean_squared_error: 15972195328.000
0 - val_loss: 20263018496.0000 - val_mean_squared_error: 20263018496.0000
Epoch 3/201
425/425 [==============================] - 4s 10ms/step - loss: 14420820992.0000 - mean_squared_error: 14420820992.00
00 - val_loss: 19543795712.0000 - val_mean_squared_error: 19543795712.0000
Epoch 4/201
425/425 [==============================] - 4s 9ms/step - loss: 13749939200.0000 - mean_squared_error: 13749939200.000
0 - val_loss: 20615002112.0000 - val_mean_squared_error: 20615002112.0000
Epoch 5/201
425/425 [==============================] - 4s 10ms/step - loss: 13731866624.0000 - mean_squared_error: 13731866624.00
00 - val_loss: 20337242112.0000 - val_mean_squared_error: 20337242112.0000
Epoch 6/201
425/425 [==============================] - 4s 9ms/step - loss: 13592888320.0000 - mean_squared_error: 13592888320.000
0 - val_loss: 19835183104.0000 - val_mean_squared_error: 19835183104.0000
Epoch 7/201
425/425 [==============================] - 4s 9ms/step - loss: 13567021056.0000 - mean_squared_error: 13567021056.000
0 - val_loss: 18868307968.0000 - val_mean_squared_error: 18868307968.0000
Epoch 8/201
425/425 [==============================] - 4s 9ms/step - loss: 13605045248.0000 - mean_squared_error: 13605045248.000
0 - val_loss: 18987311104.0000 - val_mean_squared_error: 18987311104.0000
Epoch 9/201
425/425 [==============================] - 4s 9ms/step - loss: 13607408640.0000 - mean_squared_error: 13607408640.000
0 - val_loss: 19396284416.0000 - val_mean_squared_error: 19396284416.0000
Epoch 10/201
425/425 [==============================] - 4s 9ms/step - loss: 13540501504.0000 - mean_squared_error: 13540501504.000
0 - val_loss: 19202590720.0000 - val_mean_squared_error: 19202590720.0000
Epoch 11/201
425/425 [==============================] - 4s 9ms/step - loss: 13461937152.0000 - mean_squared_error: 13461937152.000
0 - val_loss: 19572611072.0000 - val_mean_squared_error: 19572611072.0000
Epoch 12/201
425/425 [==============================] - 3s 8ms/step - loss: 13345097728.0000 - mean_squared_error: 13345097728.000
0 - val_loss: 18728927232.0000 - val_mean_squared_error: 18728927232.0000
Epoch 13/201
425/425 [==============================] - 3s 8ms/step - loss: 13428012032.0000 - mean_squared_error: 13428012032.000
0 - val_loss: 18622887936.0000 - val_mean_squared_error: 18622887936.0000
Epoch 14/201
425/425 [==============================] - 4s 9ms/step - loss: 13503724544.0000 - mean_squared_error: 13503724544.000
0 - val_loss: 19357835264.0000 - val_mean_squared_error: 19357835264.0000
Epoch 15/201
425/425 [==============================] - 4s 9ms/step - loss: 13381402624.0000 - mean_squared_error: 13381402624.000
0 - val_loss: 20509771776.0000 - val_mean_squared_error: 20509771776.0000
Epoch 16/201
425/425 [==============================] - 4s 9ms/step - loss: 13356270592.0000 - mean_squared_error: 13356270592.000
0 - val_loss: 19940605952.0000 - val_mean_squared_error: 19940605952.0000
Epoch 17/201
425/425 [==============================] - 4s 9ms/step - loss: 13435109376.0000 - mean_squared_error: 13435109376.000
0 - val_loss: 19154219008.0000 - val_mean_squared_error: 19154219008.0000
Epoch 18/201
425/425 [==============================] - 4s 9ms/step - loss: 13371157504.0000 - mean_squared_error: 13371157504.000
0 - val_loss: 19453159424.0000 - val_mean_squared_error: 19453159424.0000
Epoch 19/201
425/425 [==============================] - 4s 9ms/step - loss: 13515923456.0000 - mean_squared_error: 13515923456.000
0 - val_loss: 19156140032.0000 - val_mean_squared_error: 19156140032.0000
Epoch 20/201
425/425 [==============================] - 4s 9ms/step - loss: 13270878208.0000 - mean_squared_error: 13270878208.000
0 - val_loss: 19825807360.0000 - val_mean_squared_error: 19825807360.0000
Epoch 21/201
425/425 [==============================] - 4s 9ms/step - loss: 13375424512.0000 - mean_squared_error: 13375424512.000
0 - val_loss: 19208886272.0000 - val_mean_squared_error: 19208886272.0000
Epoch 22/201
425/425 [==============================] - 4s 9ms/step - loss: 13346455552.0000 - mean_squared_error: 13346455552.000
0 - val_loss: 19135809536.0000 - val_mean_squared_error: 19135809536.0000
Epoch 23/201
425/425 [==============================] - 4s 9ms/step - loss: 13394557952.0000 - mean_squared_error: 13394557952.000
0 - val_loss: 19593304064.0000 - val_mean_squared_error: 19593304064.0000
```

In [14]:
```python
# weight adjustment due to dropout
WEIGHTS = model_NN_light.get_weights()
WEIGHTS[0] *= 1/0.4
model_NN_light.set_weights(WEIGHTS)
```

In [15]:
```python
df_train.head(2)
```

Out[15]:

	population	median_house_value
0	1015.0	66900.0
1	1129.0	80100.0

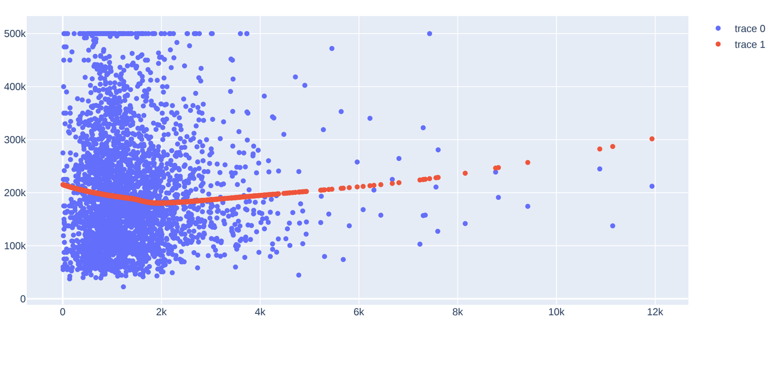
In [16]:
```python
predictions = model_NN_light.predict(X_test_norm)
```

In [17]:
```python
df_test["predictions"] = predictions
```

In [18]:
```python
fig = go.Figure()

fig.add_trace(go.Scatter(x=df_test.population, y=df_test.median_house_value, mode="markers"))
fig.add_trace(go.Scatter(x=df_test.population, y=df_test.predictions, mode="markers"))

fig.show()
```



Heavyweight Model (Regular)

In [19]:
```python
model_NN_heavy = Sequential([InputLayer(input_shape=(1,)),
            Dropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(2000, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(2000, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            Dropout(0.4),
            Dense(1, activation=None)
            ])
print(model_NN_heavy)
```

```
<keras.engine.sequential.Sequential object at 0x00000215E871DEE0>
```

In [20]:
```python
model_NN_heavy.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_5 (Dropout)          (None, 1)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 1000)              2000
_________________________________________________________________
dropout_6 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_6 (Dense)              (None, 1000)              1001000
_________________________________________________________________
dropout_7 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_7 (Dense)              (None, 2000)              2002000
_________________________________________________________________
dropout_8 (Dropout)          (None, 2000)              0
_________________________________________________________________
dense_8 (Dense)              (None, 2000)              4002000
_________________________________________________________________
dropout_9 (Dropout)          (None, 2000)              0
_________________________________________________________________
dense_9 (Dense)              (None, 1000)              2001000
_________________________________________________________________
dropout_10 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 1001
=================================================================
Total params: 9,009,001
Trainable params: 9,009,001
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```python
model_NN_heavy.compile(loss="mean_squared_error",
              optimizer="Nadam",
              metrics=["mean_squared_error"])
```

In [22]:
```python
history_NN_heavy = model_NN_heavy.fit(X_train_norm, y_train, epochs=201, validation_split=0.2, callbacks=[callback])
```

```
Epoch 1/201
425/425 [==============================] - 5s 10ms/step - loss: 17656913920.0000 - mean_squared_error: 17656913920.00
00 - val_loss: 19352459264.0000 - val_mean_squared_error: 19352459264.0000
Epoch 2/201
425/425 [==============================] - 4s 10ms/step - loss: 13510360064.0000 - mean_squared_error: 13510360064.00
00 - val_loss: 16837670912.0000 - val_mean_squared_error: 16837670912.0000
Epoch 3/201
425/425 [==============================] - 4s 10ms/step - loss: 13283505152.0000 - mean_squared_error: 13283505152.00
00 - val_loss: 18379104256.0000 - val_mean_squared_error: 18379104256.0000
Epoch 4/201
425/425 [==============================] - 4s 10ms/step - loss: 13286567936.0000 - mean_squared_error: 13286567936.00
00 - val_loss: 19538284544.0000 - val_mean_squared_error: 19538284544.0000
Epoch 5/201
425/425 [==============================] - 4s 10ms/step - loss: 13315142656.0000 - mean_squared_error: 13315142656.00
00 - val_loss: 18128340992.0000 - val_mean_squared_error: 18128340992.0000
Epoch 6/201
425/425 [==============================] - 4s 10ms/step - loss: 13172221952.0000 - mean_squared_error: 13172221952.00
00 - val_loss: 20198012928.0000 - val_mean_squared_error: 20198012928.0000
Epoch 7/201
425/425 [==============================] - 4s 10ms/step - loss: 13028730880.0000 - mean_squared_error: 13028730880.00
00 - val_loss: 18516449280.0000 - val_mean_squared_error: 18516449280.0000
Epoch 8/201
425/425 [==============================] - 4s 10ms/step - loss: 12976603136.0000 - mean_squared_error: 12976603136.00
00 - val_loss: 19205980160.0000 - val_mean_squared_error: 19205980160.0000
Epoch 9/201
425/425 [==============================] - 4s 10ms/step - loss: 13025759232.0000 - mean_squared_error: 13025759232.00
00 - val_loss: 19905568768.0000 - val_mean_squared_error: 19905568768.0000
Epoch 10/201
425/425 [==============================] - 4s 10ms/step - loss: 13117812736.0000 - mean_squared_error: 13117812736.00
00 - val_loss: 19178070016.0000 - val_mean_squared_error: 19178070016.0000
Epoch 11/201
425/425 [==============================] - 4s 10ms/step - loss: 13212876800.0000 - mean_squared_error: 13212876800.00
00 - val_loss: 20039665664.0000 - val_mean_squared_error: 20039665664.0000
Epoch 12/201
425/425 [==============================] - 4s 10ms/step - loss: 13049270272.0000 - mean_squared_error: 13049270272.00
00 - val_loss: 18354200576.0000 - val_mean_squared_error: 18354200576.0000
```

In [23]:
```python
# weight adjustment due to dropout
WEIGHTS = model_NN_heavy.get_weights()
WEIGHTS[0] *= 1/0.4
model_NN_heavy.set_weights(WEIGHTS)
```

In [24]:
```python
df_train.head(2)
```

Out[24]:

	population	median_house_value
0	1015.0	66900.0
1	1129.0	80100.0

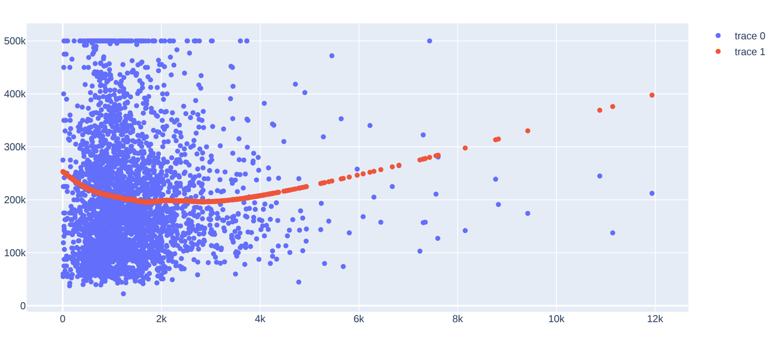
In [25]:
```python
predictions = model_NN_heavy.predict(X_test_norm)
```

In [26]:
```python
df_test["predictions"] = predictions
```

In [27]:
```python
fig = go.Figure()

fig.add_trace(go.Scatter(x=df_test.population, y=df_test.median_house_value, mode="markers"))
fig.add_trace(go.Scatter(x=df_test.population, y=df_test.predictions, mode="markers"))

fig.show()
```



Using Monte Carlo Dropout

We simply overwrite the method to turn the dropout technique on in the testing phase as well

In [28]:
```python
class MCDropout(Dropout):
    def call(self, inputs):
        return super().call(inputs, training=True)
```

Lightweight Model (MC)

In [29]:
```python
model_MC_light = Sequential([InputLayer(input_shape=(1,)),
            MCDropout(0.4),
            Dense(100, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(200, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(200, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(100, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(1, activation=None)
            ])
```

In [30]:
```python
model_MC_light.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mc_dropout (MCDropout)       (None, 1)                 0
_________________________________________________________________
dense_11 (Dense)             (None, 100)               200
_________________________________________________________________
mc_dropout_1 (MCDropout)     (None, 100)               0
_________________________________________________________________
dense_12 (Dense)             (None, 200)               20200
_________________________________________________________________
mc_dropout_2 (MCDropout)     (None, 200)               0
_________________________________________________________________
dense_13 (Dense)             (None, 200)               40200
_________________________________________________________________
mc_dropout_3 (MCDropout)     (None, 200)               0
_________________________________________________________________
dense_14 (Dense)             (None, 100)               20100
_________________________________________________________________
mc_dropout_4 (MCDropout)     (None, 100)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 101
=================================================================
Total params: 80,801
Trainable params: 80,801
Non-trainable params: 0
_________________________________________________________________
```

In [31]:
```python
model_MC_light.compile(loss="mean_squared_error",
              optimizer="Nadam",
              metrics=["mean_squared_error"])
```

In [32]:
```python
history_MC_light = model_MC_light.fit(X_train_norm, y_train, epochs=200, validation_split=0.2, callbacks=[callback])
```

```
Epoch 1/200
425/425 [==============================] - 4s 8ms/step - loss: 27748853760.0000 - mean_squared_error: 27748853760.000
0 - val_loss: 25316454400.0000 - val_mean_squared_error: 25316454400.0000
Epoch 2/200
425/425 [==============================] - 4s 9ms/step - loss: 14788104192.0000 - mean_squared_error: 14788104192.000
0 - val_loss: 21069961216.0000 - val_mean_squared_error: 21069961216.0000
Epoch 3/200
425/425 [==============================] - 4s 9ms/step - loss: 13878960128.0000 - mean_squared_error: 13878960128.000
0 - val_loss: 20582518784.0000 - val_mean_squared_error: 20582518784.0000
Epoch 4/200
425/425 [==============================] - 3s 8ms/step - loss: 13518361600.0000 - mean_squared_error: 13518361600.000
0 - val_loss: 20375926784.0000 - val_mean_squared_error: 20375926784.0000
Epoch 5/200
425/425 [==============================] - 4s 9ms/step - loss: 13569750016.0000 - mean_squared_error: 13569750016.000
0 - val_loss: 21484328960.0000 - val_mean_squared_error: 21484328960.0000
Epoch 6/200
425/425 [==============================] - 4s 10ms/step - loss: 13635933184.0000 - mean_squared_error: 13635933184.00
00 - val_loss: 20539156480.0000 - val_mean_squared_error: 20539156480.0000
Epoch 7/200
425/425 [==============================] - 4s 9ms/step - loss: 13579567104.0000 - mean_squared_error: 13579567104.000
0 - val_loss: 20155623424.0000 - val_mean_squared_error: 20155623424.0000
Epoch 8/200
425/425 [==============================] - 4s 9ms/step - loss: 13662499840.0000 - mean_squared_error: 13662499840.000
0 - val_loss: 21012750336.0000 - val_mean_squared_error: 21012750336.0000
Epoch 9/200
425/425 [==============================] - 4s 9ms/step - loss: 13548626944.0000 - mean_squared_error: 13548626944.000
0 - val_loss: 20307726336.0000 - val_mean_squared_error: 20307726336.0000
Epoch 10/200
425/425 [==============================] - 4s 9ms/step - loss: 13585741824.0000 - mean_squared_error: 13585741824.000
0 - val_loss: 20109516800.0000 - val_mean_squared_error: 20109516800.0000
Epoch 11/200
425/425 [==============================] - 4s 9ms/step - loss: 13418787840.0000 - mean_squared_error: 13418787840.000
0 - val_loss: 20180346880.0000 - val_mean_squared_error: 20180346880.0000
Epoch 12/200
425/425 [==============================] - 4s 9ms/step - loss: 13575069696.0000 - mean_squared_error: 13575069696.000
0 - val_loss: 20755804160.0000 - val_mean_squared_error: 20755804160.0000
Epoch 13/200
425/425 [==============================] - 4s 9ms/step - loss: 13521255424.0000 - mean_squared_error: 13521255424.000
0 - val_loss: 20398833664.0000 - val_mean_squared_error: 20398833664.0000
Epoch 14/200
425/425 [==============================] - 3s 8ms/step - loss: 13420542976.0000 - mean_squared_error: 13420542976.000
0 - val_loss: 20926963712.0000 - val_mean_squared_error: 20926963712.0000
Epoch 15/200
425/425 [==============================] - 4s 9ms/step - loss: 13486310400.0000 - mean_squared_error: 13486310400.000
0 - val_loss: 19677827072.0000 - val_mean_squared_error: 19677827072.0000
Epoch 16/200
425/425 [==============================] - 4s 9ms/step - loss: 13412325376.0000 - mean_squared_error: 13412325376.000
0 - val_loss: 20243664896.0000 - val_mean_squared_error: 20243664896.0000
Epoch 17/200
425/425 [==============================] - 4s 9ms/step - loss: 13502494720.0000 - mean_squared_error: 13502494720.000
0 - val_loss: 20359710720.0000 - val_mean_squared_error: 20359710720.0000
Epoch 18/200
425/425 [==============================] - 4s 9ms/step - loss: 13447757824.0000 - mean_squared_error: 13447757824.000
0 - val_loss: 20931121152.0000 - val_mean_squared_error: 20931121152.0000
Epoch 19/200
425/425 [==============================] - 3s 8ms/step - loss: 13547769856.0000 - mean_squared_error: 13547769856.000
0 - val_loss: 20362485760.0000 - val_mean_squared_error: 20362485760.0000
Epoch 20/200
425/425 [==============================] - 4s 8ms/step - loss: 13392892928.0000 - mean_squared_error: 13392892928.000
0 - val_loss: 21054955520.0000 - val_mean_squared_error: 21054955520.0000
Epoch 21/200
425/425 [==============================] - 4s 8ms/step - loss: 13468219392.0000 - mean_squared_error: 13468219392.000
0 - val_loss: 20182489088.0000 - val_mean_squared_error: 20182489088.0000
Epoch 22/200
425/425 [==============================] - 4s 9ms/step - loss: 13460331520.0000 - mean_squared_error: 13460331520.000
0 - val_loss: 21070665728.0000 - val_mean_squared_error: 21070665728.0000
Epoch 23/200
425/425 [==============================] - 4s 9ms/step - loss: 13485915136.0000 - mean_squared_error: 13485915136.000
0 - val_loss: 21077284864.0000 - val_mean_squared_error: 21077284864.0000
Epoch 24/200
425/425 [==============================] - 4s 9ms/step - loss: 13337717760.0000 - mean_squared_error: 13337717760.000
0 - val_loss: 20511672320.0000 - val_mean_squared_error: 20511672320.0000
Epoch 25/200
425/425 [==============================] - 4s 10ms/step - loss: 13436920832.0000 - mean_squared_error: 13436920832.00
00 - val_loss: 20489043968.0000 - val_mean_squared_error: 20489043968.0000
```

In [33]:
```python
# weight adjustment due to dropout
WEIGHTS = model_MC_light.get_weights()
WEIGHTS[0] *= 1/0.4
model_MC_light.set_weights(WEIGHTS)
```

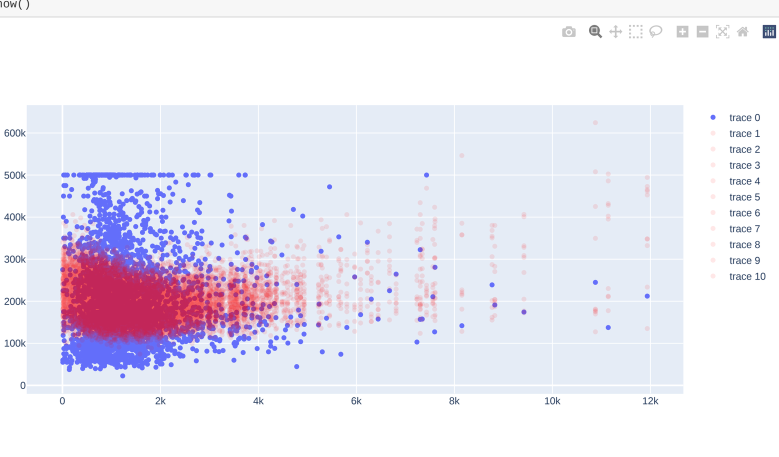
In [34]:
```python
y_pred_stack = np.stack([model_MC_light.predict(X_test_norm) for sample in range(10)])
```

In [35]:
```python
fig = go.Figure()

fig.add_trace(go.Scatter(x=df_test.population, y=df_test.median_house_value, mode="markers"))
for i in range(len(y_pred_stack)):

    fig.add_trace(go.Scatter(x=df_test.population, y=y_pred_stack[i].reshape(-1,), mode="markers", opacity=0.1,  mar
ker=dict(
            color='Red')))

fig.show()
```



The plot is quite spread and wide

Heaveweight Model (MC)

In [36]:
```python
model_MC_heavy = Sequential([InputLayer(input_shape=(1,)),
            MCDropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(2000, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(2000, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(1000, activation="relu", kernel_initializer="lecun_normal"),
            MCDropout(0.4),
            Dense(1, activation=None)
            ])
```

In [37]:
```python
model_MC_heavy.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mc_dropout_5 (MCDropout)     (None, 1)                 0
_________________________________________________________________
dense_16 (Dense)             (None, 1000)              2000
_________________________________________________________________
mc_dropout_6 (MCDropout)     (None, 1000)              0
_________________________________________________________________
dense_17 (Dense)             (None, 1000)              1001000
_________________________________________________________________
mc_dropout_7 (MCDropout)     (None, 1000)              0
_________________________________________________________________
dense_18 (Dense)             (None, 2000)              2002000
_________________________________________________________________
mc_dropout_8 (MCDropout)     (None, 2000)              0
_________________________________________________________________
dense_19 (Dense)             (None, 2000)              4002000
_________________________________________________________________
mc_dropout_9 (MCDropout)     (None, 2000)              0
_________________________________________________________________
dense_20 (Dense)             (None, 1000)              2001000
_________________________________________________________________
mc_dropout_10 (MCDropout)    (None, 1000)              0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 1001
=================================================================
Total params: 9,009,001
Trainable params: 9,009,001
Non-trainable params: 0
_________________________________________________________________
```

In [38]:
```python
model_MC_heavy.compile(loss="mean_squared_error",
              optimizer="Nadam",
              metrics=["mean_squared_error"])
```

In [39]:
```python
history_MC_heavy = model_MC_heavy.fit(X_train_norm, y_train, epochs=200, validation_split=0.2, callbacks=[callback])
```

```
Epoch 1/200
425/425 [==============================] - 5s 11ms/step - loss: 17611278336.0000 - mean_squared_error: 17611278336.00
00 - val_loss: 17869496320.0000 - val_mean_squared_error: 17869496320.0000
Epoch 2/200
425/425 [==============================] - 4s 10ms/step - loss: 13449282560.0000 - mean_squared_error: 13449282560.00
00 - val_loss: 19761350656.0000 - val_mean_squared_error: 19761350656.0000
Epoch 3/200
425/425 [==============================] - 4s 10ms/step - loss: 13282202624.0000 - mean_squared_error: 13282202624.00
00 - val_loss: 21133776896.0000 - val_mean_squared_error: 21133776896.0000
Epoch 4/200
425/425 [==============================] - 4s 10ms/step - loss: 13318580224.0000 - mean_squared_error: 13318580224.00
00 - val_loss: 20077113344.0000 - val_mean_squared_error: 20077113344.0000
Epoch 5/200
425/425 [==============================] - 4s 10ms/step - loss: 13161405440.0000 - mean_squared_error: 13161405440.00
00 - val_loss: 19969955840.0000 - val_mean_squared_error: 19969955840.0000
Epoch 6/200
425/425 [==============================] - 4s 10ms/step - loss: 13174627328.0000 - mean_squared_error: 13174627328.00
00 - val_loss: 21065703424.0000 - val_mean_squared_error: 21065703424.0000
Epoch 7/200
425/425 [==============================] - 4s 10ms/step - loss: 13109524480.0000 - mean_squared_error: 13109524480.00
00 - val_loss: 20322779136.0000 - val_mean_squared_error: 20322779136.0000
Epoch 8/200
425/425 [==============================] - 4s 10ms/step - loss: 13099629568.0000 - mean_squared_error: 13099629568.00
00 - val_loss: 21221609472.0000 - val_mean_squared_error: 21221609472.0000
Epoch 9/200
425/425 [==============================] - 4s 10ms/step - loss: 13099910144.0000 - mean_squared_error: 13099910144.00
00 - val_loss: 22886950912.0000 - val_mean_squared_error: 22886950912.0000
Epoch 10/200
425/425 [==============================] - 4s 10ms/step - loss: 13024440320.0000 - mean_squared_error: 13024440320.00
00 - val_loss: 19406856192.0000 - val_mean_squared_error: 19406856192.0000
Epoch 11/200
425/425 [==============================] - 4s 10ms/step - loss: 13248636928.0000 - mean_squared_error: 13248636928.00
00 - val_loss: 20529242112.0000 - val_mean_squared_error: 20529242112.0000
```

In [40]:
```python
# weight adjustment due to dropout
WEIGHTS = model_MC_heavy.get_weights()
WEIGHTS[0] *= 1/0.4
model_MC_heavy.set_weights(WEIGHTS)
```

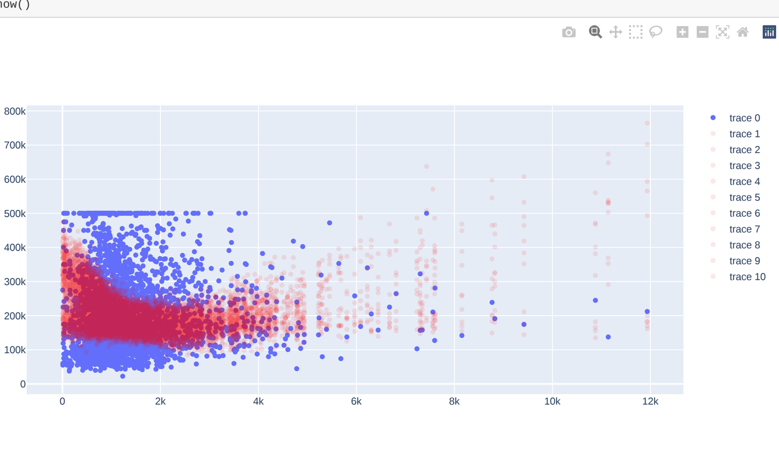
In [41]:
```python
y_pred_stack = np.stack([model_MC_heavy.predict(X_test_norm) for sample in range(10)])
```

In [42]:
```python
fig = go.Figure()

fig.add_trace(go.Scatter(x=df_test.population, y=df_test.median_house_value, mode="markers"))
for i in range(len(y_pred_stack)):

    fig.add_trace(go.Scatter(x=df_test.population, y=y_pred_stack[i].reshape(-1,), mode="markers", opacity=0.1,  mar
ker=dict(
            color='Red')))

fig.show()
```



The plot for the heavyweight model is more packed than the previous one, which shows that the model is less uncertain and more confident than the previous model